LOAD BALANCING TECHNIQUES FOR I/O INTENSIVE TASKS ON HETEROGENEOUS CLUSTERS

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology

in

Computer Science & Engineering

By

Sukromony Lakra

Department Of Computer Science & Engineering

National Institute Of Technology

Rourkela

2007

LOAD BALANCING TECHNIQUES FOR I/O INTENSIVE TASKS ON HETEROGENEOUS CLUSTERS

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology

in

Computer Science & Engineering

By

Sukromony Lakra

Under the Guidance of

BibhuDatta Sahoo

Department Of Computer Science & Engineering
National Institute Of Technology
Rourkela

2007

National Institute Of Technology

Rourkela

CERTIFICATE

This to certify that the thesis entitled “Load Balancing Techniques for I/O Intensive Tasks on Heterogeneous Clusters ”submitted by Ms. Sukromony Lakra in partial fulfillment of the requirements for the award of Master of Technology degree in Computer Science Engineering with specialization in ” Computer Science ” at the National Institute of Technology, Rourkela(Deemed University) is an authentic work carried out by her under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other university/institute for the award of any degree/diploma.

Bibhudatta Sahoo
Senior Lecturer
Dept. of Computer Science & Engg.
National Institute of Technology
Rourkela - 769008

Date:

Acknowledgement

I wish to express my heartiest thanks to all who extended their unlimited help to me during my thesis work and its subsequent documentation.

I wish to express my sincere gratitude to my guide, Sr. Lecturer. B.D.Sahoo, for his kind and able guidance for the completion of this thesis work. His consistent support and intellectual guidance made me energize and innovate new ideas.

I am grateful to Dr. S. K. Jena, Head of the Department, Computer Science Engineering, NIT Rourkela for his support during my work.

I am thankful to all professors and lecturers and members of the department of Computer Science and Engineering, NIT, Rourkela for their generous help in various ways for the completion of the thesis work.

I would like to thank my classmates at NIT, Rourkela for their generous help in various ways which resulted in the completion of the Documentation.

Sukromony Lakra
Roll No: 20506004
M.Tech., 4^{th} Semester
Computer Science & Engg.

Contents

Abstract

Load balancing schemes in a cluster system play a critically important role in developing high-performance cluster computing platform. Existing load balancing approaches are concerned with the effective usage of CPU and memory resources. I/O-intensive tasks running on a heterogeneous cluster need a highly effective usage of global I/O resources, previous CPU-or memory-centric load balancing schemes suffer significant performance drop under I/O-intensive workload due to the imbalance of I/O load. To solve this problem, Zhang et al. developed two I/O-aware load-balancing schemes, which consider system heterogeneity and migrate more I/O-intensive tasks from a node with high I/O utilization to those with low I/O utilization. If the workload is memory-intensive in nature, the new method applies a memory-based load balancing policy to assign the tasks. Likewise, when the workload becomes CPU-intensive, their scheme leverages a CPU-based policy as an efficient means to balance the system load. In doing so, the proposed approach maintains the same level of performance as the existing schemes when I/O load is low or well balanced. Results from a trace-driven simulation study show that, when a workload is I/O-intensive, the proposed schemes improve the performance with respect to mean slowdown over the existing schemes by up to a factor of 8. In addition, the slowdowns of almost all the policies increase consistently with the system heterogeneity.

List of Figures

List of Tables

Chapter 1

CLUSTER

1.1 INTRODUCTION

Computer cluster is a collection of workstations or PCs that are interconnected by a high-speed network working together as a single, integrated computing resource.A single PC or node is a single or multiprocessor system with memory, I/O facilities, & OS.Generally two or more computers (nodes) connected together in a single cabinet, or physically separated & connected via a LAN.It appears as a single system to users and applications.

A cluster is a type of parallel or distributed processing system, which consists of a collection of interconnected stand-alone computers cooperatively working together as a single, integrated computing resource.

A computer cluster is a group of loosely coupled computers that work together closely so that in many respects they can be viewed as though they are a single computer.The components of a Cluster are commonly, but not always, connected to each other through fast local area networks. Clusters are usually deployed to improve speed and/or reliability over that provided by a single computer, while typically being much more cost-effective than single computers of comparable speed or reliability.

A cluster typically consists of:

- Compute nodes
- Storage

- Interconnection networks

Main characteristic features of clusters

- A cluster connects complete computers
- The component computers of a cluster are loosely connected
- A cluster is utilized as a single, unified computing resource: Single System Image (like supercomputers)

1.2 CLUSTER CLASSIFICATION

1.2.1 Application Target

High-availability(HA) clusters

High-availability clusters are implemented primarily for the purpose of improving the availability of services which the cluster provides. They operate by having redundant nodes, which are then used to provide service when system components fail. The most common size for an HA cluster is two nodes, which is the minimum required to provide redundancy. HA cluster implementations attempt to manage the redundancy inherent in a cluster to eliminate single points of failure.

There are many commercial implementations of High-Availability clusters for many operating systems. The Linux-HA project is one commonly used free software HA package for the Linux OS.

Load-balancing clusters

Load-balancing clusters operate by having all workload come through one or more load-balancing front ends, which then distribute it to a collection of back end servers. Although they are implemented primarily for improved performance, they commonly include high-availability features as well. Such a cluster of computers is sometimes referred to as a server farm. There are many commercial load balancers available including Platform LSF HPC, Sun Grid Engine, Moab Cluster Suite and Maui Cluster Scheduler. The Linux Virtual Server project provides one commonly used free software package for the Linux OS.

High-performance(HPC) clusters

High-performance clusters are implemented primarily to provide increased performance by splitting a computational task across many different nodes in the cluster, and are most commonly used in scientific computing. One of the most popular HPC implementations is a cluster with nodes running Linux as the OS and free software to implement the parallelism. This configuration is often referred to as a Beowulf cluster. Such clusters commonly run custom programs which have been designed to exploit the parallelism available on HPC clusters. Many such programs use libraries such as MPI which are specially designed for writing scientific applications for HPC computers.

HPC clusters are optimized for workloads which require jobs or processes happening on the separate cluster computer nodes to communicate actively during the computation. These include computations where intermediate results from one node's calculations will affect future calculations on other nodes.

1.2.2 Node Ownership

- Dedicated Clusters(Supercomputer)
  - goal: high performance
  - method: parallel computing
  - ownership: joint
- Non-Dedicated Clusters(NOW)
  - goal: usage of spare computing cycles
  - method: background job distribution
  - ownership: individual owners of workstations

1.2.3 Node Hardware

- Clusters of PCs
- Clusters of Workstations
- Clusters of SMPs

1.2.4 Node Operating System

- Linux(Beowulf)
- Microsoft NT(Illinois HPVM)
- SUN Solaris(Berkeley NOW)
- IBM AIX(IBM SP2)
- HP UX(Illinois-PANDA)
- Mach (Microkernel based OS)(CMU)
- Cluster Operating Systems(Solaris MC,SCO Unixware,MOSIX (academic project))

1.2.5 Node Configuration

- Homogeneous Clusters

  All nodes will have similar architectures and running same OSs.Every processor is exactly like every other in capability, resources, software, and communication speed.Example : cluster of SMP machines, Beowulf cluster etc.

- Heterogeneous Clusters

  Nodes based on different processors and running different OSs.Different types of heterogeneity - architecture,data format, computational speed, system software, machine load, network load, etc.

1.2.6 Levels of Clustering

- Group Clusters(2-99 nodes)
- Departmental Clusters(10-100 nodes)
- Organizational Clusters(many 100s nodes)
- National Metacomputers(WAN/Internet based)
- International Metacomputers(Internet based,1000s to million computers)

1.2.7 Processors

- Intel: Pentiums, Xeon
- Sun: SPARC, ULTRASPARC
- HP PA
- IBM RS6000/PowerPC
- SGI MPIS
- Digital Alphas

1.2.8 High Performance Networks

- Ethernet(10Mbps)
- Fast Ethernet(100Mbps)
- Gigabit Ethernet(1Gbps)
- Myrinet
- ATM
- FDDI

Myrinet

- full duplex interconnection network
- Use low latency cut-through routing switches, which is able to offer fault tolerance by automatic mapping of the network configuration
- Support both Linux & NT
- **Advantages:**
  - very low latency
  - very high throughput
- **Disadvantages:**
  - Expensive: $1500 per host
  - Complicated scaling: switches with more than 16 ports are unavailable

1.2.9 Network Interfaces

- Network Interface Card
  - Myrinet has NIC
  - User-level access support
  - Alpha 21364 processor integrates processing, memory controller, network interface into a single chip.

1.2.10 Communication Software

- Active Messages(Berkeley)
- Fast Messages(Illinois)
- U-net(Cornell)
- XTP(Virginia)

1.2.11 Middleware Components

- Hardware

  DEC Memory Channel, DSM (Alewife,DASH) SMP Techniques
- OS/Gluing Layers

  Solaris MC, Unixware, Glunix
- Applications and Subsystems

  System management and electronic forms,Runtime systems(software DSM,PFS etc.),Resource management and scheduling(RMS):CODINE,LSF,PBS,NQS,etc.

1.2.12 Middleware and SSI

- Cluster supported by a middleware layer that resides between the OS and user-level environment.
- A single system image is the illusion, created by software or hardware, that presents a collection of resources as one, more powerful resource.
- SSI makes the cluster appear like a single machine to the user, to applications, and to the network.A cluster without a SSI is not a cluster.

SSI Benifits:

- Provide a simple, straightforward view of all system resources and activities, from any node of the cluster.
- Free the end user from having to know where an application will run.
- Free the operator from having to know where a resource is located.
- Let the user work with familiar interface and commands and allows the administrators to manage the entire clusters as a single entity.
- Reduce the risk of operator errors, with the result that end users see improved reliability and higher availability of the system.

1.2.13 Programming environments

- Threads (PCs, SMPs) POSIX Threads,Java Threads
- MPI(Message Passing Interface)
- PVM

1.2.14 Development Tools

- Compiler(C,C++,Java)
- RAD (rapid application development tools).. GUI based tools for PP modeling
- Debuggers
- Performance Analysis Tools
- Visualization Tools

1.2.15 Applications

- Sequential
- Parallel or Distributed(Cluster-aware application.)
  - Weather Forecasting
  - Quantum Chemistry

- Molecular Biology Modeling
- Engineering Analysis (CAD/CAM)
- Web servers,Data-mining

1.2.16 Benefits of Clustering

- High Performance
- Expandability and Scalability
- High Throughput
- High Availability

1.3 Popularity of clusters

Low cost,scalimg, and fault isolation proved a perfect match to the companies providing services over the Internet since the mid-1990s. Internet applications such as search engines and email serversre amenable to more loosely coupled computers, since the parallelism consists of millions of independent tasks.Hence, companies like Amazon, AOL, Google, Hotmail, Inktomi, WebTV, and Yahoo rely on clusters of PCs or workstations to provide services used by millions of people every day.Clusters are growing in popularity in the scientific computing market as well.

1.4 Cluster Architecture

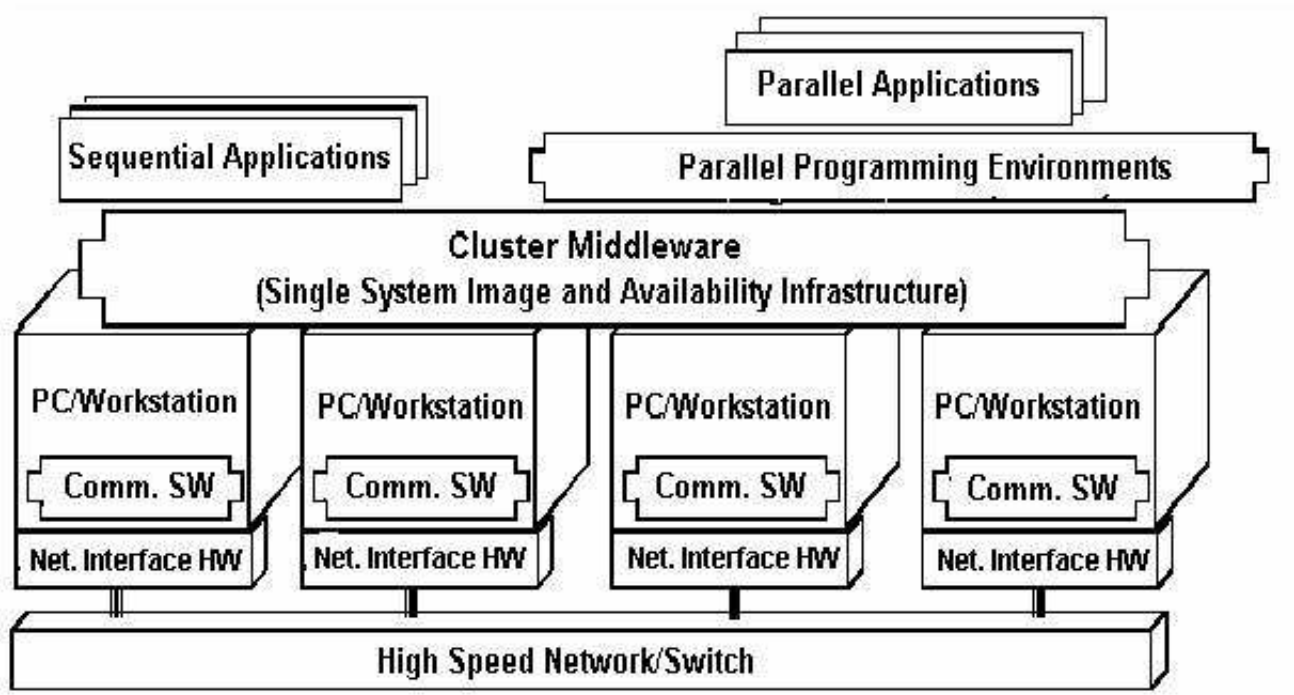

Figure 1.1: Cluster Architecture

1.5 FLYNN'S CLASSIFICATION

Flynn's taxonomy distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of Instruction and Data. Each of these dimensions can have only one of two possible states: Single or Multiple.
The matrix below defines the 4 possible classifications according to Flynn:

SISD Uni-processors	**SIMD** Processor arrays pipelined vector Processor
MISD Systolic array	**MIMD** Multiprocessor Multicomputer

Figure 1.2: Flynn's classification

1.5.1 SISD

- A serial (non-parallel) computer.
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle.
- Single data: only one data stream is being used as input during any one clock cycle.
- Deterministic execution
- This is the oldest and until recently, the most prevalent form of computer
- Examples: most PCs, single CPU workstations and mainframes

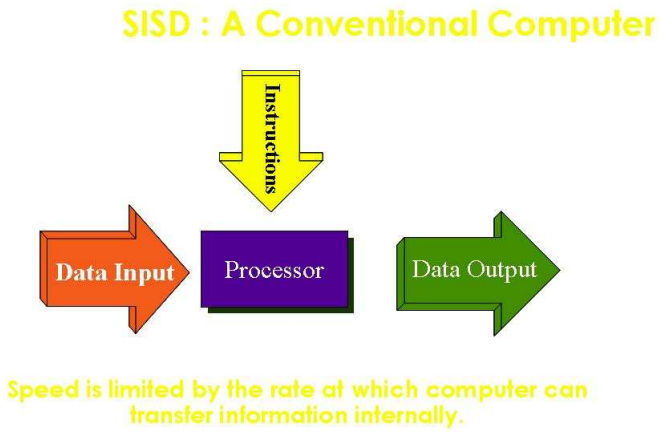

Figure 1.3: SISD

1.5.2 SIMD

- A type of parallel computer
- Single instruction: All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit can operate on a different data element
- This type of machine typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units
- Best suited for specialized problems characterized by a high degree of regularity,such as image processing.
- Synchronous (lockstep) and deterministic execution
- Two varieties: Processor Arrays and Vector Pipelines
- Examples: Processor Arrays: Connection Machine CM-2, Maspar MP-1, MP-2 Vector Pipelines: IBM 9000, Cray C90, Fujitsu VP, NEC SX-2, Hitachi S820

1.5.3 MISD

- A single data stream is fed into multiple processing units.

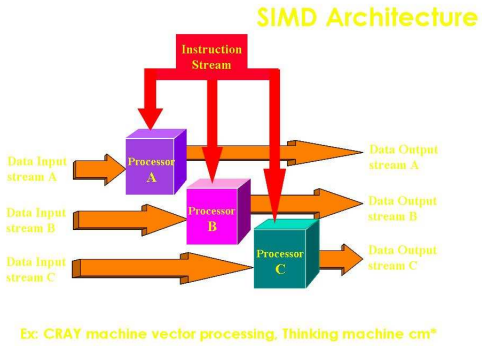

Figure 1.4: SIMD

- Each processing unit operates on the data independently via independent instruction streams.
- Few actual examples of this class of parallel computer have ever existed. One is the experimental Carnegie-Mellon C.mmp computer (1971).
- Some conceivable uses might be: multiple frequency filters operating on a single signal stream multiple cryptography algorithms attempting to crack a single coded message.

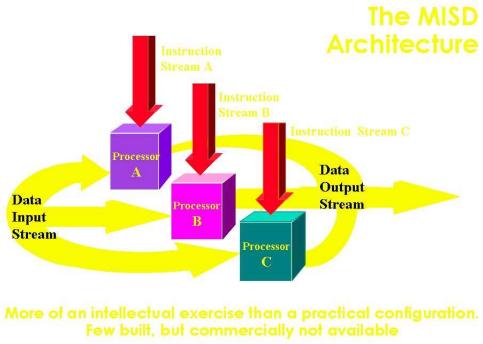

Figure 1.5: MISD

1.5.4 MIMD

- Currently, the most common type of parallel computer. Most modern computers fall into this category.
- Multiple Instruction: every processor may be executing a different instruction stream
- Multiple Data: every processor may be working with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Examples: most current supercomputers, networked parallel computer, grids and multi-processor SMP computers(Clusters) including some types of PCs.

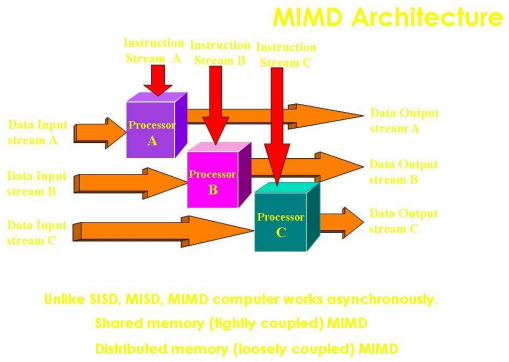

Figure 1.6: MIMD

- MIMD Shared Memory
  - Shared memory parallel computers vary widely, but generally have in common the ability for all processors to access all memory as global address space.
  - Multiple processors can operate independently but share the same memory resources.
  - Changes in a memory location effected by one processor are visible to all other processors.
  - Shared memory machines can be divided into two main classes based upon memory access times: UMA and NUMA.

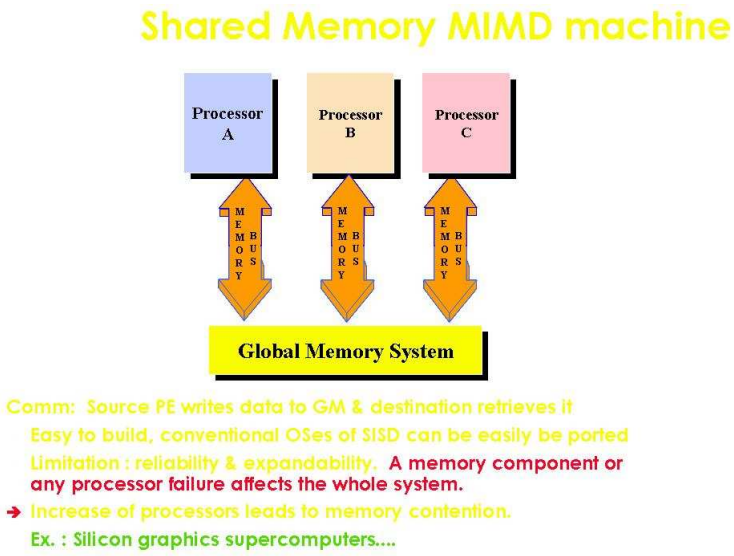

Figure 1.7: MIMD Shared Memory

- MIMD Distributed Memory
  - Like shared memory systems, distributed memory systems vary widely but share a common characteristic. Distributed memory systems require a communication network to connect inter-processor memory.

- Processors have their own local memory. Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processors.
- Because each processor has its own local memory, it operates independently. Changes it makes to its local memory have no effect on the memory of other processors. Hence, the concept of cache coherency does not apply.
- When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated. Synchronization between tasks is likewise the programmer's responsibility.
- The network,fabric, used for data transfer varies widely, though it can can be as simple as Ethernet.

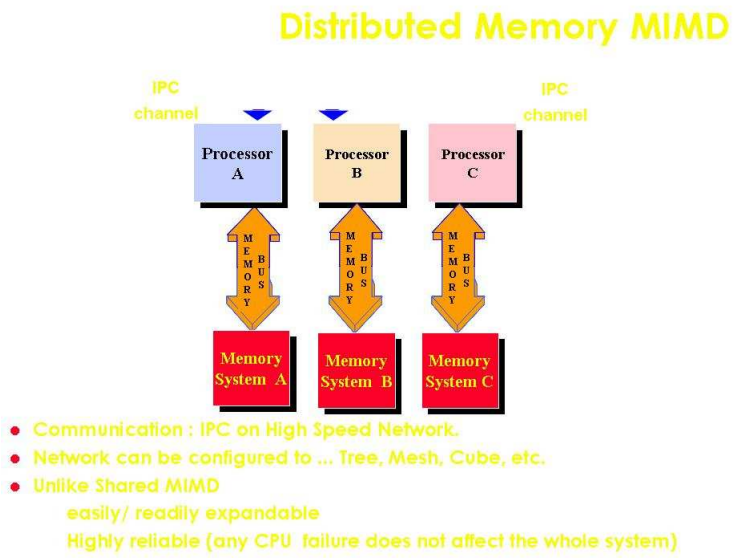

Figure 1.8: MIMD Distributed Memory

Chapter 2

LOAD BALANCING

2.1 LOAD

A cluster consists of number of nodes, and each node has a combination of multiple types of resources, such as CPU, memory, network connectivity and disks.In a node resource queue lengths and particularly the CPU queue length are good indicators of load because they correlate well with the task response time.Except CPU load there are two more loads,they are:Memory load and I/O load.

2.2 Types of Load

- CPU load

  CPU load of a node is characterized by the length of the CPU waiting queue.

- Memory load

  Memory load of a node is the sum of the memory space allocated to all the tasks running on that node.

- I/O load

  I/O load measures two types of I/O accesses, i.e

  - Implicit I/O requests includes by page fault
  - explicit I/O requests issued from tasks.

2.3 Load Balancing

A clustered system can be load balanced which is the act of distributing a computer's workload among other members of the cluster to ensure the work is completed quickly and that one device is not doing all the work .In a cluster system, balancing the load among the nodes is very very necessary such that the overall performance of the system is maximized.A load manager resides in each node is responsible for load balancing and available resources of the node.All load manager in a cluster is capable of keeping track of global load informatioon by monitoring local resources and sharing load information through the communication network.

Load manager balances the load in a node using certain load balancing algorithms or policies.The aim of the load balancing policies is to equally distribute the load of the system among all the nodes.

On a network of shared processors, load balancing is the idea of migrating processes or tasks across the network from hosts with high loads to hosts with lower loads.The motivation for load balancing is to reduce the average completion time of processes and improve the utilization of the processors.

An important part of the load-balancing strategy is the migration policy, which determines when migrations occur and which tasks are migrated.

Task migration for purposes of load balancing comes in two forms: remote execution (also called non-preemptive migration), in which some new tasks are (possibly automatically) executed on remote hosts, and preemptive migration, in which running tasks may be suspended, moved to a remote host, and restarted.

2.3.1 Preemptive vs. Nonpreemptive migration

Preemptive task migration involves the migration of a task that is partially executed.This migration is an expensive operationas the collection of task's state can be difficult.Typically, a task state consists of a virtual memory image, a process control block, unread I/O buffers and messages, file pointers, timers that have been set, etc.

Nonpreemptive task migration, on the other hand, involve the migration of tasks that have not begun execution and hence do not require the migration of the task's state.In both types of migration, information about the environment in which the task will execute must be transferred to the recieving node. This information can include user's current working directory, the privileges inherited by the task, etc.Nonpreemptive task migration is also reffered to as *task placements*.

Load balancing may be done explicitly (by the user) or implicitly (by the system). Implicit migration policies may or may not use a priori information about the function of tasks, how long they will run, etc. If the cost of remote execution is significant relative to the lifetimes of tasks, then implicit non-preemptive policies require some a priori information about job lifetimes. This in formation is often implemented as an eligibility list that specifies which task may be migrated.

In contrast, most preemptive migration policies do not use a priori information, since this it is often difficult to maintain and preemptive strategies can perform well without it. These systems use only system-visible data like the current age of each task and its memory size.

Preemptive migration performs better than non-preemptive migration.

2.3.2 Load Balancing vs. Load Sharing

Load distributing algorithms can be further be classified as *load balancing* or *load sharing* algorithms, based on their load distributing principle.Both types of algorithms strive to reduce the likelihood of an *unshared state* (a state in which one computer lies idle while at the same time tasks contend for service at another computer) by transferring tasks to lightly loaded nodes.Load balancing algorithms, however, go to a step further by attempting to equalize loads at all computers.Because a load balancing algorithm transfer tasks at a higher rate than a load sharing algorithm, the higher overhead incurred by the load balancing algorithm may outweigh this potential performance improvement.

2.3.3 Load balancing Classifications

In heterogeneous systems, the processing power may vary from one site to another, and also the jobs may arrive unevenly at the different sites in the system;

this entails that some sites are temporarily overloaded while others idle or underloaded.

Load balancing plays a central role in system utilization by almost equalizing the loads on the system. As a result the situation where some sites are heavily loaded and others might be idle is avoided. Many load balancing mechanisms have been developed, and many approaches for classifying these methods also were introduced.One of the classifications of load balancing is based on the time of activating the load balancer; this class has two types:

- Static load balancing
- Dynamic load balancing

In static load balancing schemes, assignment of tasks to processors is done compilation time before tasks are executed, usually using a priori knowledge about the tasks and system on which they run.A main advantage of these techniques is that they will not introduce any run-time overhead.It is fixed throughout the execution time.The static algorithm runs periodically.

In dynamic load balancing, no decision is made until tasks start executing in the system. The scheme uses tasks and system state information when making the load balancing decisions. Dynamic algorithms are especially critical in applications where parameters that affect the scheme cannot be easily determined a priori or when the workload evolves as computation progresses.These load distribution algorithms incur run-time overhead.Dynamic load balancing schemes periodically assigns tasks as needed during execution to achieve balance.

2.4 Key issues to consider when designing a dynamic load balancing algorithm are:

- Transfer policy
- Selection policy
- Information policy
- Location policy

2.4.1 Transfer policy

A transfer policy that determines whether a node is in a suitable state to participate in a task transfer.A large no of the transfer policies that have been proposed are *threshold* policies.Thresholds are expressed in units of load.When a new task originates at a node, and the load at that node exceeds a threshold T, the transfer policy decides that the node is a *sender*.If the load at a node falls below T, the transfer policy decides thay the node can be *reciever* for a remote task.

An alternative transfer policy initiates task transfers whenever an imbalance in load among nodes is detected because of the actions of the information policy.

2.4.2 Selection policy

A selection policy selects a task for transfer,once the transfer policy decides that the node is a sender.Shold the selection policy fail to find a suitable task to transfer, the node is no longer considered a sender until the transfer policy decides that the node is a sender again.

The simplest approach is to select newly originated tasks that have caused the node to become a sender by increasing the load at the node beyond the threshold.Such tasks are relatively cheap to transfer, as the transfer in nonpreemptive.

A basic criterion that a task selected for transfer should satisfy is that the overhead incurred in the transfer of the task shold be compensated for by the reduction in the response time realized by the task.In general, long-lived tasks satisfy this criterion.Also, a task can be selected for remote execution if the estimated average execution time for that type of task is greater than some execution time threshold.A task is selected for transfer only if its response time will be improved upon transfer.

There are other factors to consider in the selection of a task.First, the overhead incurred by the transfer should be minimal.Foe example, a task of small size carries less overhead.Second, the no of location dependent system calls made by the selected task should be minimal.Location-dependent calls must be executed at the node where the task originated because they use resources such as windows, or the mouse, that only exist at the node.

2.4.3 Location policy

The responsibility of a location policy is to find suitable nodes (senders or recievers) to share load. A widely used method for finding a suitable node is through *polling*.In polling, a node polls another node to find out whether it is suitable node for load sharing.Nodes can be polled either serially or in parallel. A node can be selected for polling either randomly, based on the information collected during the previous polls, or on a nearest-neighbor basis.An alternative to polling is to broadcast a query to find out if any node is available for load sharing.

2.4.4 Information policy

The information policy is responsible for deciding when information about the states of other nodes in the system should be collected, where it should be collected from, and what information should be collected.Most information policies are one of the following three types:

- Demand-driven
- Periodic
- State-change-driven

Demand-driven

In this class of policy, a node collects the state of other nodes only when it becomes either a sender or a reciever (decided by the transfer and selection policies at the node), making it a suitable candidate to initiate load sharing.Note that a demand-driven information policy is inherently a dynamic policy, as its actions depend on the system state. Demand-driven policies can be *sender-initiated*, *reciever-initiated*, or *symmetrically initiated*. In sender-initiated policies, sender look for recievers to transfer their load. In reciever-initiated policies, recievers solicit load from senders. a symmetrically initiated policy is a combination of both, where load sharing actions are triggered by the demand for extra processing power or extra work.

Periodic

In this class of policy, nodes exchange load information periodically.Based on the information collected, the transfer policy at a node may decide to transfer jobs.

Periodic information policies do not adapt their activity to the system state.for example, the benefits due to load distributing are minimal at high sytem loads because most of the nodes in the system are busy. Nevertheless, overheads due to periodic information collection continue to increase the sytem load and thus worsen the situation.

State-change-driven

In this class of policy, nodes disseminate state information whenever their state changes by a certain degree. A state-change-driven policy differs from a demand-driven policy in that it disseminates information about the state of a node, rather than collecting information about other nodes. Under centralized state-change-driven policies, nodes send state information to a centralized colection point.Under decentralized state-change-driven policies, nodes send information to peers.

Dynamic load balancing strategies can also be classified as:

- Centralized
- Distributed

Centralized load balancing algorithms have a master node which allocates tasks to the other processors thus maintaining a balanced workload.

In distributed algorithms, each processor makes makes an independent load balancing decision based on a combination of its own load and system-wide load information.

Another classification considers the process of load balancing as a job routing problem. The clustered systems are represented by appropriate queuing models, and these algorithms are classified depending on the sites that initiate the process of load balancing, they are:

- Sender initiated
- Reciever Initiated

In Sender-Initiated algorithm, heavily loaded node (sender) initiates transfer of task to lightly loaded node (reciever).

In Reciever-Initiated schemes, lightly loaded node or under loaded node (reciever) trying to obtain a task from an overloaded node (sender).

Load balancing also can be classified according to the properties of location policy into three categories : deterministic, probabilistic, and dynamic.Both deterministic and probabilistic approaches do not use the state information, and thus, cannot react to a dynamic situation. The load balancing algorithms are further divided into several levels according to the amount of information required for them.

2.5 Load balancing Algorithms

There are following three load balancing algorithms:

- Random Load Balancing Algorithm
- Diffusion Algorithm
- Complete redistribution

2.5.1 Random Load Balancing Algorithm

The random strategy is based on a simple heuristics.A processor decides to send some of its tasks to another processor if the no of tasks assigned to it is greater than a certain threshold no of tasks.

2.5.2 Diffusion Algorithm

The diffusion algorithm is analogous to the physical process of diffusion where tasks flow between processors with excess tasks diffusing to neighbouring processors that have fewer tasks.The diffusion algorithm applied to a system with uneven load distribution and identical processors will eventually result in a balanced load distribution.each processoe examines the task average of allits neighbor processors and sends out tasks if its load is greater than a certain threshold, a function of the local load information.It sends out tasks to all neighbors that have load less than the average local load.

2.5.3 Complete redistribution

The complete redistribution algorithm requires complete (global) state information.The algorithm is activated if the load on any of the processors exceeds the threshold.

2.6 Dynamic Load balancing Algorithms

Generally, dynamic load-balancing algorithms take take following three approaches:

- Least-loaded approach
- Threshold-based approach
- Bidding approach

2.6.1 Least-loaded approach

The least-loaded approach attempts to allocate tasks to the least-loaded computers in the system.

2.6.2 Threshold-based approach

In the threshold-based approach, a workstation triggers load-balancing actions if its load level exceeds a certain threshold.Threshold-based algorithms take one of three approaches. In the sender-initiated approach, the highly loaded computers dispatch their tasks to computers with lighter loads. In the receiver-initiated approach, the lightly loaded computers request tasks from the busy computers. The symmetrically initiated approach combines these two schemes.

2.6.3 Bidding approach

The bidding approach views the computers as resources and the tasks as consumers. For example, the Spawn system simulates an open financial market, using a form of priority for currency.Application managers bid for CPU time; only the winners can execute jobs on workstations. However, because the bidding process takes nonnegligible time, it might not work well in the LAN environment, which consists of both short and long jobs with unknown characteristics.

Chapter 3

TASK

3.1 Definition

A batch system executes *jobs*, whereas a time-shared system has *user programs*, or *tasks*.Task is also reffered as *process*.A process is a program in execution.A process is the unit of wok in a modern time-sharing system.

3.2 Task State

As a task executes, it changes state. The state of a task is defined by the current activity of that task.Each task may be in one of the following state:

- New : The task is being created.
- Running : Instructions are being executed.
- Waiting : The task is waiting for some event to occur (suach as an I/O completion).
- Ready : The task is waiting to be assigned to a processor.
- Terminated : The task has finished execution.

3.3 Types of Tasks

In general, most tass can be described as following three types:

- CPU-Intensive

- Memory-Intensive
- I/O-Intensive

An I/O-intensive task spends more of its time doing I/O than it spends doing computations. A CPU-Intensive task, on the other hand, generates I/O request infrequently, using more of its time doing computation rather than an I/O-Intensive task uses.A Memory-intensive task performs only memory operations.

Each task is described by its requirements for CPU, memory, amd I/O, which are measured by Seconds, Mbytes, and no of I/O accesses per ms, respectively.Job with intensive I/O requests can be regarded as having two sub-tasks, namely, computational task along with the CPU and memory demands, and I/O task asociated with I/O requirement.

3.4 Task Scheduling Algorithms

Tasks arrive at each node dynamically and independently, and share resources available there. Each node maintains its individual task queue where newly arrived tasks may be transmitted to other nodes or executed in the local node, depending on a load balancing policy employed in the system. Each node keeps track of reasonably up-to-date global load information by periodically exchanging load status with other nodes.

Before executing in a particular node, the task is allocated to a processor for execution using following scheduling algorithms:

3.4.1 Scheduling Algorithms

- First-Come, First-Served (FCFS)
- Shortest-Job-First (SJF)
- Priority Scheduling
- Round-Robin (RR)

3.5 Non-Preemptive Vs. Preemptive Scheduling

3.5.1 Non-Preemptive

Non-preemptive algorithms are designed so that once a process/task enters the running state(is allowed a process), it is not removed from the processor until it has completed its service time (or it explicitly yields the processor).

3.5.2 Preemptive

Preemptive algorithms are driven by the notion of prioritized computation. The process with the highest priority should always be the one currently using the processor. If a process is currently using the processor and a new process with a higher priority enters, the ready list, the process on the processor should be removed and returned to the ready list until it is once again the highest-priority process in the system.

3.6 FCFS

This is a Non-Premptive scheduling algorithm. FCFS strategy assigns priority to task in the order in which they request the processor.The task that requests the CPU first is allocated the CPU first.When a task comes in, it is added to the tail of task queue. When running task terminates, dequeue the task at head of task queue and run it.While the FCFS algorithm is easy to implement, it ignores the service time request and all other criteria that may influence the performance with respect to turnaround or waiting time.

3.7 SJF

Maintain the task queue in order of increasing job lengths. When a job comes in, insert it in the task queue based on its length. When current task is done, pick the one at the head of the queue and run it.SJF is proven optimal only when all jobs are available simultaneously.

3.8 Priority Scheduling

Run highest-priority task first, use round-robin among tasks of equal priority. Re-insert process in task queue behind all tasks of greater or equal priority.Equal-priority tasks are scheduled in FCFS order.Priority scheduling may cause low-priority processes to starve.

3.9 Round-Robin

The Round-Robin schedulingalgorithm is designed especially for time-sharing systems.It is similar to FCFS scheduling, but preemption is added to switch between tasks.A small unit of time, called a time quantum (or time slice), is defined.A time quantum is generally from 10 to 100 milliseconds.The task queue is treated as circular queue.The scheduler goes around the task queue, allocating to each task for a time interval of upto 1 time quantum.

Chapter 4

LOAD BALANCING TECHNIQUES FOR I/O INTENSIVE TASKS ON HETEROGENEOUS CLUSTER

4.1 Related work

A cluster consists of a number of nodes, and each node has a combination of multiple types of resources, such as CPU, memory, network connectivity and disks. In a cluster system, dynamic load balancing schemes can improve system performance by attempting to assign work, at run time, to machines with idle or under-utilized resources.

Load balancing schemes are widely recognized as important techniques for the efficient utilization of resources in networks of workstations or clusters.Many load balancing polices achieve high system performance by increasing the utilization of CPU [13, 4], memory [1, 14], or a combination of CPU and memory [9, 10, 8, 16].Although these load-balancing policies have been very effective in increasing the utilization of resources in distributed systems, they have ignored disk I/O, which is a likely performance bottleneck when a large number of applications running on clusters are data-intensive and/or I/O-intensive. Therefore, we believe that for any dynamic load balancing scheme to be effective in this new application environment, it must be made I/Oaware.Typical examples of I/O-intensive applications include long running simulations of time-dependent phenomena that periodically generate snapshots of their state [19], archiving of raw and processed remote sensing data [3, 15], multimedia and web-based applica-

tions.These applications share a common feature in that their storage and computational requirements are extremely high. Therefore, the high performance of I/O-intensive applications heavily depends on the effective usage of storage, in addition to that of CPU and memory. Compounding the performance impact of I/O in general, and disk I/O in particular, is the steadily widening gap between CPU and I/O speed, making the load imbalance in I/O increasingly more crucial to overall system performance. To bridge this gap,I/O buffers allocated in the main memory have been successfully used to reduce disk I/O costs, thus improving the throughput of I/O systems. In this regard, load balancing with I/O-awareness, when appropriately designed, is potentially capable of boosting the utilization of the I/O buffer in each node, which in turn increases the buffer hit rate and decreases disk I/O access frequency.

Zhang et al. proposed two I/O-aware load-balancing schemes to improve overall performance of a distributed system with a general and practical workload including I/O activities [11, 22].However, it is assumed in [11, 22] that the system is homogeneous in nature.There is a strong likelihood that upgraded clusters or networked clusters are heterogeneous, and heterogeneity in disks tends to induce more significant performance degradation when coupled with imbalanced load of memory and I/O resources.Therefore, it becomes imperative that heterogeneous clusters be capable of hiding the heterogeneity of resources, especially that of I/O resources, by judiciously balancing I/O work across all the nodes in a cluster.This paper studies two dynamic load balancing policies, which are shown to be effective for improving the utilization of disk I/O resources in heterogeneous clusters.

There is a large body of literature concerning load balancing in disk systems. Scheuermann et al. [14] have studied the issues of striping and load balancing in parallel disk systems.To minimize total I/O time in heterogeneous cluster environments, Cho et al. [27] have developed heuristics to choose the number of I/O servers and place them on physical processors.In [25, 26], Qin et al. have studied dynamic scheduling algorithms to improve the read and write performance of a parallel file system by balancing the global workload. The above techniques can improve system performance by fully utilizing the available hard drives. However, these approaches become less effective under a complex workload where I/O-intensive tasks share resources with many memory- and CPU-intensive tasks.

Many researchers have shown that I/O cache and buffer are useful mechanisms

to optimize storage systems. Ma et al. [21] have implemented active buffering to alleviate the I/O burden by using local idle memory and overlap- ping I/O with computation.Qin et al. have developed a feedback control mechanism to improve the performance of a cluster by manipulating the I/O buffer size [23].Forney et al. [2] have investigated storage-aware caching algorithms in heterogeneous clusters. Although we focus solely on balancing disk I/O load in this paper, the approach proposed here is capable of improving the buffer utilization of each node. The scheme presented in this paper is complementary to the existing caching/buffering techniques, thereby providing additional performance improvement when combined with active buffering, storage-aware caching, and a feedback control mechanism.

4.2 Workload and System Model

A cluster computing platform considered in this study consists of a set N=N1,N2,.....,Nn of n heterogeneous nodes connected by a high-speed network.Tasks arrive at each node dynamically and independently, and share resources available there. Each node maintains its individual task queue where newly arrived tasks may be transmitted to other nodes or executed in the local node, depending on a load balancing policy employed in the system. Each node keeps track of reasonably up-to-date global load information by periodically exchanging load status with other nodes.

Architecture of a Cluster System

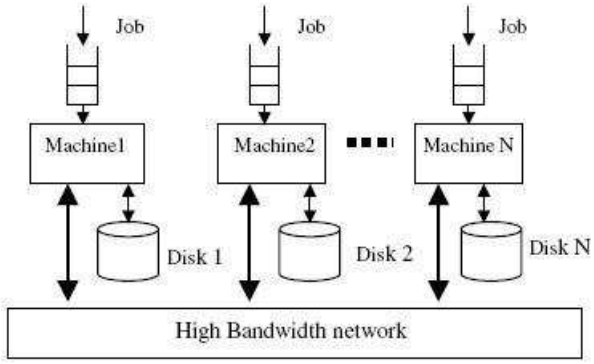

Figure 4.1: Architecture of a Cluster System

Heterogeneity only refers to the variations of CPU powers, memory capacities and disk capacity, but not the variations of operating systems, network interfaces and hardware organizations among the workstations.Let the cluster consisted of n heterogeneous nodes.Each node i is characterize by its CPU speed C_i,, memory capacity M_i, and disk performance D_i.Let B_i^{disk} , S_i, and R_i denote the disk bandwidth,average seek time, and average rotation time of the disk in node i, then the disk performance can be approximately measured as the following equation:

$$D_i = \frac{1}{S_i + R_i + d/B_i^{disk}}$$

where d is the average size of data stored or retrieved by I/O requests.

The weight of a disk performance W_i^{disk} is depended as a ratio between its performance and that of the fastest disk in the cluster. Thus, we have

$$W_i^{disk} = \frac{D_i}{\max_{i=1}^{n} D_j}$$

.
The disk heterogeneity level, referred to as H_D, can be quantitatively measured by the standard deviation of disk weights. Thus, H_D is expressed as:

$$H_D = \sqrt{\frac{\sum_{i=1}^{n} (W_{avg}^{disk} - W_i^{disk})^2}{n}}$$

where W_{avg}^{disk} is the average disk weight. Likewise, the CPU and memory heterogeneity levels are defined as follows:

$$H_C = \sqrt{\frac{\sum_{i=1}^{n} (W_{avg}^{CPU} - W_i^{CPU})^2}{n}}$$

$$H_M = \sqrt{\frac{\sum_{i=1}^{n} (W_{avg}^{mem} - W_i^{mem})^2}{n}}$$

where W_i^{CPU} and W_i^{mem} are the CPU and memory weights, and W_{avg}^{CPU} and W_{avg}^{mem} are the average weights of CPU and memory resources.

4.2.1 Task

Each task has the following five major parameters:

1. Arrival node
2. Arival time
3. Requested memory size
4. Duration time
5. Disk access rate

4.3 Load Balancing in Heterogeneous Clusters

4.3.1 Existing Load Balancing Policies

In principle, the load of a node can be balanced by migrating either a newly arrived job or a currently running job preemptively to another node if needed.While the first approach is referred to as Remote Execution, the second one is called preemptive migration . Since the focuses of this study are effective usage of I/O resources and coping with system heterogeneity, only the technique of remote execution is considered.In what follows, review of two existing load-balancing policies.

CPU-based Load Balancing CPU-RE [12, 5]

We consider a simple and effective policy, which is based on a heuristic. The CPU load of node i,$load_{CPU}(i)$,is measured by the following expression:
$load_{CPU}(i) = L_i \times \frac{max_{j=1}^{n} C_j}{C_i}$ where L_i is the number of tasks on node i.

If $load_{CPU}(i)$ is greater than a certain threshold, node i is consider overloaded with respect to CPU resource. The CPU-based policy transfers the newly arrived

tasks on the overloaded node i to the remote node with the lightest CPU load. This policy is capable of resulting in a balanced CPU load distribution for systems with uneven CPU load distribution [12]. Note that the CPU threshold is a key parameter that depends on both workload and transfer cost. In [9], the value of CPU threshold is set to four.

Weakness of CPU-RE Load Balancing Policy

A major performance objective of implementing a load sharing policy in a distributed system is to minimize execution time of each individual job, and to maximize the system throughput by effectively using the distributed resources, such as CPUs, memory modules, and I/Os. Most load sharing schemes (e.g., [13, 4, 7, 20, 6, 18, 17]) mainly consider CPU load balancing by assuming each computer node in the system has a sufficient amount of memory space. These schemes have proved to be effective on overall performance improvement of distributed systems. However, with the rapid development of CPU chips and the increasing demand of data accesses in applications, the memory resources in a distributed system become more and more expensive relative to CPU cycles. Zhang et al. believed that the overheads of data accesses and movement, such as page faults, have grown to the point where the overall performance of distributed systems would be considerably degraded without serious considerations concerning memory resources in the design of load sharing policies. We have following reasons to support our claim. First, with the rapid development of RISC and VLSI technology, the speed of processors has increased dramatically in the past decade. We have seen an increasing gap in speed between processor and memory, and this gap makes performance of application programs on uniprocessor, multiprocessor and distributed systems rely more and more on effective usage of their entire memory hierarchies. In addition, the memory and I/O components have a dominant portion in the total cost of a computer system. Second, the demand for data accesses in applications running on distributed systems has significantly increased accordingly with the rapid establishment of local and wide-area internet infrastructure. Third, the latency of a memory miss or a page fault is about 1000 times higher than that of a memory hit.Therefore, minimizing page faults through memory load sharing has a great potential to significantly improve overall performance of distributed systems. Finally, it has been shown that memory utilizations among the different nodes in a distributed system are highly unbalanced in practice, where page faults frequently occur in some heavily loaded nodes but a few memory accesses or no memory

accesses are requested on some lightly loaded nodes or idle nodes [1]. So Zhang et al. designed new load sharing policy to share both CPU and memory services among the nodes in order to minimize both CPU idle times and the number of page faults in distributed systems.

CPU-memory-based Load Balancing CM-RE.[9, 10, 8, 16]

This policy takes both CPU and memory resources into account. The memory load of node i, $load_{mem}(i)$, is the sum of the memory space allocated to the tasks running on node i. Thus: $load_{mem}(i) = \sum_{j \in N_i} mem(j)$ where mem(j) represents the memory requirement of task j, and N_i denotes the set of tasks running on node i.

If the memory space of a node is greater than or equal to $load_mem(i)$, CM-RE adopts the above CPU-RE policy to make load-balancing decisions. When $load_mem(i)$ exceeds the amount of available memory space, the CM-RE policy transfers the newly arrived tasks on the overloaded node to the remote nodes with the lightest memory load.Zhang et al. [9] showed that CM-RE is superiors to CPU-RE under a memory-intensive workload.

A load sharing policy considering only CPU or only memory resource would be beneficial either to CPU-bound or to memory-bound jobs. Only a load sharing policy considering both resources will be beneficial to jobs of both types.

4.4 IO-aware Load Balancing in Heterogeneous Clusters

I/O-aware load balancing policy (IO-RE) [24] is heuristic and greedy in nature. Instead of using CPU and memory load indices, the IO-RE policy relies on an I/O load index to measure two types of I/O access: the implicit I/O load induced by page faults and the explicit I/O requests resulting from tasks accessing disks. Let $page(i, j)$ be the implicit I/O load of task j on node i, and $IO(j)$ be the explicit I/O requirement of task j. Thus, node i's I/O load index is:

$$load_{IO}(i) = \sum_{j \in N_i} page(i, j) + \sum_{j \in N_i} IO(j)$$

An I/O threshold, $threshold_{IO}(i)$, is introduced to identify whether node i's I/O

resource is overloaded. Node i's I/O resource is considered overloaded if $load_{IO}(i)$ is higher than $threshold_{IO}(i)$. Specifically, $threshold_{IO}(i)$, which reflects the I/O processing capability of node i, is expressed as:

$$threshold_{IO}(i) = \frac{D_i}{\sum_{j=1}^{n} D_j} \times \sum_{j=1}^{n} load_{IO}(j)$$

where the first term on the right hand side of the above equation corresponds to the fraction of the total I/O processing power on node i, and the second term gives the accumulative I/O load imposed by the running tasks in the heterogeneous cluster.

For a task j arriving at a local node i, the IO-RE scheme attempts to balance I/O resources in the following four main steps. First, the I/O load of node i is updated by adding task j's explicit and implicit I/O load. Second, the I/O threshold of node i is computed. Third, if node i's I/O resource is underloaded, task j will be executed locally on node i. When the node is overloaded with respect to I/O resource, IO-RE judiciously chooses a remote node k as task j's destination node, subject to the following two conditions: The I/O resource is not overloaded. The I/O load discrepancy between node i and k is greater than the I/O load induced by task j, to avoid useless migrations. If such a remote node is not available, task j has to be executed locally on node i. Otherwise and finally, task j is transferred to the remote node k, and the I/O load of nodes i and k is updated in accordance with j's load.

When the available memory space M_i is unable to fulfill the accumulative memory requirements of all tasks running on the node $load_{mem}(i)$ is greater than M_i, the node may encounter a large number of page faults. Implicit I/O load depends on three factors: M_i, $load_{mem}(i)$, and the page fault rate pr_i. Thus, $page(i, j)$ can be depended as follows:

$$page(i,j) = \begin{cases} 0 & \text{if } load_{mem}(i) \leq M_i \\ \frac{pr_i \times load_{mem}(i)}{M_i} & \text{otehrwise} \end{cases}$$

Explicit I/O load $IO(i,j)$ is proportional to I/O access rate $ar(j)$ and inversely proportional to I/O buffer hit rate $hr(i,j)$. The buffer hit rate for task j on node i is approximated by the following formula:

$$hr(i,j) = \begin{cases} \frac{r}{r+1} & \text{if } buf(i,j) \geqslant d(j) \\ \frac{r \times buf(i,j)}{(r+1) \times d(j)} & \text{otehrwise} \end{cases}$$

where r is the data re-access rate (defined to be the number of times the same data is accessed by a task), $buf(i,j)$ denotes the buffer size allocated to task j, and $d(j)$ is the amount of data accessed by task j, given a buffer with infinite size. The buffer size a task can obtain at run time heavily depends on the total available buffer size in the node and the tasks' access patterns. $d(j)$ is linearly proportional to access rate, computation time and average data size of I/O accesses, and $d(j)$ is inversely proportional to r. In some cases, where the initial data of a task j is not initially available at the remote node, the data migration overhead can be approximately estimated as

$$Td(j) = \frac{d_{init}(j)}{b_{net}}$$

,where $d_{init}(j)$ and b_{net} represent the initial data size and the available network bandwidth, respectively.

4.5 IOCM-RE: A Comprehensive Load Balancing Policy

Since the main target of the IO-RE policy is exclusively I/O-intensive workload,IO-RE is unable to maintain a high performance when the workload tends to be CPU- or memory-intensive. To overcome this limitation of IO-RE, a new approach, referred to as IOCM-RE, attempts to achieve the effective usage of CPU and memory in addition to I/O resources in heterogeneous clusters.

More specifically, when the explicit I/O load of a node is greater than zero,the I/O-based policy will be leveraged by IOCM-RE as an efficient means to make load-balancing decisions. When the node exhibits no explicit I/O load, either the

memory-based or the CPU-based policy will be utilized to balance the system load. In other words, if the node has implicit I/O load due to page faults, load-balancing decisions are made by the memory-based policy. On the other hand, the CPU-based policy is used when the node is able to fulfill the accumulative memory requirements of all tasks running on it. A pseudo code of the IOCM-RE scheme is presented in Figure below.

Algorithm: IO-CPU-Memory based load balancing (IOCM-RE):
/* Assume that a task j newly arrives at node i */
if $IO(j) + \sum_{j \in N_i} IO(k) > 0$ then
  The IO-RE policy is used to balance the system node; /* see Section 3.2 */
else if $page(i,j) + \sum_{j \in N_i} page(i,k) > 0$ then /* see Section 3.1(2) */
    The memory-based policy is utilized for load balancing;
  else /* see Section 3.1(1) */
    The CPU-based policy makes the load balancing decision;

Fig. 1. Pseudocode of the IO-CPU-Memory based load balancing

4.6 Performance Evaluation

Zhang et al. experimentally compared the performance of IOCM-RE against that of three other schemes: CPU-RE [12, 5], CM-RE [9], and IO-RE (Section 4.4). The performance metric by which they judge system performance is the *mean slowdown*. Formally, the mean slowdown of all the tasks in trace T is given below, where w_i and $l_C(i)$ are wall-clock execution time and computation load of task i. The implicit and explicit disk I/O load of task i are denoted as $l_{page}(i)$ and $l_{IO}(i)$, respectively.

$$slowdown(T) = \frac{\sum_{i \in T} w_i}{\sum_{i \in T}((\frac{n \times l_C(i)}{\sum_{j=1}^{n} C_j}) + (\frac{n \times (l_{page}(i) + l_{IO}(i))}{\sum_{j=1}^{n} D_j}))}$$

Note that the numerator is the summation of all the tasks' wall-clock execution time while sharing resources, and the denominator is the summation of all tasks' time spent running on CPU or accessing I/O without sharing resources with other tasks. Since the clusters are heterogeneous in nature, the average values of CPU power, memory space,and the disk performance are taken to calculate the execution time of each task exclusively running on a node.

4.7 Overall Performance Comparison

The mean slowdowns of four policies increase considerably as one of the disks ages. This is because aging one slow disk gives rise to longer I/O processing time.

I/O-aware policies ignore the heterogeneity in CPU resources. When the heterogeneity of CPU and memory remain unchanged, IO-RE and IOCM-RE is less sensitive to the change in disk I/O heterogeneity than the other three policies. This is because both IO-RE and IOCM-RE consider disk heterogeneity as well as the effective usage of I/O resources.

4.8 Simulation and Simulation Parameters

Harchol-Balter and Downey [13] have implemented a simulator of a distributed system with six nodes. Zhang et. al [9] extended the simulator by incorporating memory recourses. Compared to these two simulators, Zhang et. al [24] possesses four new features. First, the IOCM-RE and IO-RE schemes are implemented. Second, a fully connected network is simulated. Third, a simple disk model is

Parameters	*Value*
CPU Speed	100-400 MIPS
RAM Size	32-256 MByte
Buffer Size	64MByte
Context switch time	0.1 ms
Page Fault Service Time	8.1 ms
Mean page fault rate	0.01No./MI
Data re-access rate,r	5
Network Bandwidth	100Mbps

Table 4.1: Table 1. System Parameters. CPU speed and page fault rate are measured by Millions Instruction Per Second (MIPS) and No.Million Instructions (No.MI), respectively.

added into the simulator. Last, an I/O buffer is implemented on top of the disk model in each node. In all experiments, they used the simulated system with the configuration parameters listed in Table 1. The parameters are chosen in such a way that they resemble workstations such as the Sun SPARC-20.

Disk I/O operations issued by each task are modeled as a Poisson Process with a mean arrival rate λ, which is set to 2 No/MI in their experiments. The service time of each I/O access is the summation of seek time, rotation time, and transfer time. The transfer time is equal to the size of accessed data divided by the disk bandwidth. Data sizes are randomly generated based on a Gamma distribution with the mean size of 256KByte.

The configuration of disks used in their simulated environment is based on the assumption of device aging and performance-fault injection. Specifically, IBM 9LZX is chosen as a base disk and its performance is aged over years to generate a variety of disk characteristics [2], which is shown in Table 2.

4.9 Overall Performance Comparison

In this experiment, the CPU power and the memory size of each node are set to 200MIPS and 256MByte. Figure 1 plots the mean slowdown as a function of disk age. Disks are configured such that five fast disks are one year old, and a sixth,

Age	*sktime*	*Rtime*	*Bandwidth*
1yr	5.3 ms	3.00ms	20.0MB/s
2yr	5.89ms	3.33ms	14.3MB/s
3yr	6.54ms	3.69ms	10.2MB/s
4yr	7.27ms	4.11ms	7.29MB/s
5yr	8.08ms	4.56ms	5.21MB/s
6yr	8.98ms	5.07ms	3.72MB/s

Table 4.2: Table 2. Characteristics of Disk Systems. sk time: seek time, R time: Rotation time

slower disk assumed an age ranging from 1 to 6 years.

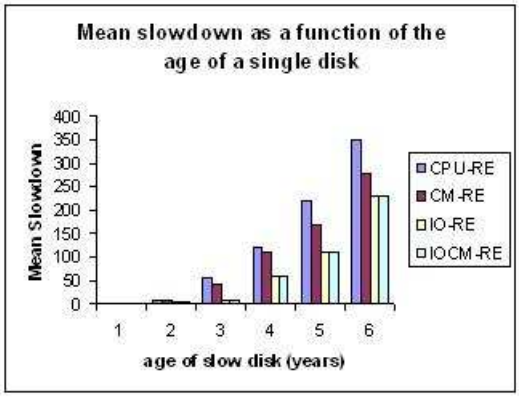

Figure 4.2: Fig.1. Mean slowdown as a function of a single disk

Figure 1 shows that the mean slowdowns of four policies increase considerably as one of the disks ages. This is because aging one slow disk gives rise to longer I/O processing time. A second observation from Figure 2 is that IO-RE and IOCM-RE perform the best out of the four policies, and they improve the performance over the other two policies by up to a factor of 8. The performance improvement of IO-RE and IOCM-RE relies on the technique that balances I/O load by migrating I/O-intensive tasks from overloaded nodes to underloaded ones.

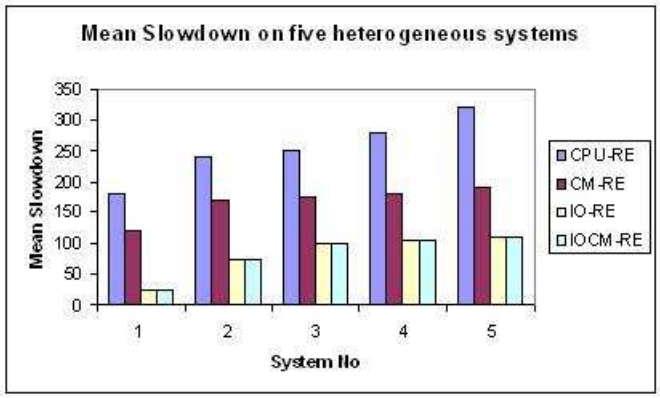

Figure 4.3: Fig.2. Mean slowdown on five heterogeneous systems.

Node	*cpu*	*mem*	*disk*	*cpu*	*mem*	*disk*	*cpu*	*mem*	*disk*	*cpu*	*mem*	*disk*	*cpu*	*mem*	*dis*
1	100	480	20	100	480	20	100	480	20	100	480	20	100	480	20
2	100	480	20	150	640	20	150	640	20	200	800	20	200	800	14
3	100	480	20	150	640	20	150	640	20	200	800	20	200	800	20
4	100	480	20	50	320	20	50	320	10.2	50	320	14.3	50	320	5.2
5	100	480	20	100	480	20	100	480	20	50	320	14.3	50	320	7.2
6	100	480	20	50	320	20	50	320	10.2	50	320	10.2	50	320	3.7
HL	0	0	0	0.27	0.20	0	0.27	0.20	0.25	0.35	0.28	0.20	0.35	0.28	0.3

Table 4.3: Table 3. Characteristics of Five Heterogeneous Clusters. CPU and memory are measured by MIPS and MByte. Disks are characterized by bandwidth measured in MByte/S. HL-Heterogeneity Level

4.10 Impact of Heterogeinity on the Performance of Load Balancing Policies

In this section, they turn their attention to the impact of system heterogeneity on the performance of the proposed policies. The five configurations of increasing heterogeneity of a heterogeneous cluster with 6 nodes are summarized in Table 3. As can be seen from Figure 2, IO-RE and IOCM-RE significantly outperform the other two policies. For example, IOCM-RE improves the performance over CPU-RE and CM-RE by up to a factor of 5 and 3, respectively.

Importantly, Figure 2 shows that the mean slowdowns of almost all policies increase consistently as the system heterogeneity increases. An interesting observation from this experiment is that the mean slowdowns of IO-RE and IOCM-RE

are more sensitive to changes in CPU and memory heterogeneity than the other two policies. Recall that system B's CPU and memory heterogeneities are higher than those of system A. Compared the performance of system A with that of B, the mean slowdowns of IO-RE and IOCM-RE are increased by 196.4%, whereas the slowdowns of CPU-RE and CM-RE are increased approximately by 34.7% and 47.9%, respectively. The reason is that I/O-aware policies ignore the heterogeneity in CPU resources. When the heterogeneity of CPU and memory remain unchanged, IO-RE and IOCM-RE is less sensitive to the change in disk I/O heterogeneity than the other three policies. This is because both IO-RE and IOCM-RE consider disk heterogeneity as well as the effective usage of I/O resources.

4.11 Conclusion

Zhang et al. have studied two I/O-aware load-balancing policies, IO-RE (I/O-based policy) and IOCM-RE (load balancing for I/O, CPU, and Memory), for heterogenous clusters executing applications that represent a diverse workload conditions, including I/O-intensive and memory-intensive applications.IOCM-RE considers both explicit and implicit I/O load, in addition to CPU and memory utilizations. To evaluate the effectiveness of their approaches, they have compared the performance of the proposed policies against two existing approaches: CPU-based policy (CPU-RE) and CPU-Memory-based policy (CM-RE). IOCM-RE is more general than the existing approaches in the sense that it can maintain high performance under diverse workload conditions. From a trace-driven simulation three empirical results are drawn:

1. When a workload is I/O-intensive, the proposed scheme improves the performance with respect to mean slowdown over the existing schemes by upto a factor of 8.

2. The slowdowns of the four policies considerably increase as one of the disks ages.

3. The slowdowns of almost all the policies increase consistently with the system heterogeneity.

Bibliography

[1] Acharya A. and Setia S. Availability and utility of idle memory in workstation clusters,. In *Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 35–46, 1999.

[2] Forney B., Arpaci-Dusseau AC., and Arpaci-Dusseau RH. Storage-aware caching:revisiting caching for heterogeneous storage systems,. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies ,Monterey, CA*. USENIX Association Berkeley, CA, USA, 2002.

[3] Chang C., Moon B., Acharya A., Shock C., Sussman A., and Saltz J. Titan: A high-performance remote-sensing database,. In *Proc. of International Conf. on Data Engineering*, 1997.

[4] Hui C. and Chanson S. Improved strategies for dynamic load balancing,. *Concurrency,IEEE*, 7(3):969–989, September 1999.

[5] Eager D., Lazowska ED., and Zahorjan J. Adaptive load sharing in homogeneous distributed systems,. *IEEE Transactions on Software Engineering*, 12(5):662–675, May 1986.

[6] Eager DL., Lazowska ED., and Zahorjan J. The limited performance benefits of migrating active processes for load sharing. In *Proceedings of ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems,*, pages 63–72, May 1998.

[7] Douglis F. and Ousterhout J. Transparent process migration: Design alternatives and the sprite implementation,. *Software Practice and Experience,*, 21(8):757–785, 1991.

[8] Xiao L., Chen S., and Zhang X. Dynamic cluster resource allocations for jobs with known and unknown memory demands,. *IEEE Transactions on Parallel and Distributed Systems*, 13(3), 2002.

[9] Xiao L., Qu Y., and Zhang X. Effective load sharing on heterogeneous networks of workstations,. In *Proceedings of 2000 International Parallel and Distributed Processing Symposium (IPDPS2000)*, May 2000.

[10] Xiao L., Qu Y., and Zhang X. Improving distributed workload performance by sharing both cpu and memory resources,. In *Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS 2000)*, pages 233–241, April 2000.

[11] Xiao L., Qu Y., and Zhang X. A dynamic load balancing scheme for i/o-intensive applications in distributed systems,. In *Proceedings of the 2003 International Conference on Parallel Processing Workshops (ICPPW03)*, pages 431–438. IEEE, 2003.

[12] Franklin M. and Govindan V. A general matrix iterative model for dynamic load balancing,. *ACM Portal,Parallel Computing*, 22(7):969–989, October 1996.

[13] Harchol-Balter M. and Downey A. Exploiting process lifetime distributions for load balancing,. *ACM Transactions on Computer Systems*, 15(3):253–285, August 1997.

[14] Scheuermann P., Weikum G., and Zabback P. Data partitioning and load balancing in parallel disk systems,. *The VLDB*, pages 48–66, 1998.

[15] Ferreira R., Moon B., Humphries J., Sussman A., Saltz J., Miller R., and Demarzo A. The virtual microscope,. In *Proc. of the 1997 AMIA Annual Fall Symposium,*, pages 449–453, Oct 1997.

[16] Chen S., Xiao L., and Zhang X. Dynamic load sharing with unknown memory demands of jobs in clusters,. In *Proc. 21 International Conf. Distributed Computing Systems (ICDCS 2001)*, April 2001.

[17] Zhou S. A trace-driven simulation study of load balancing,. *IEEE Transactions on Software Engineering,,*, 14(9):1327–1341, 1988.

[18] Kunz T. The influence of different workload descriptions on a heuristic load balancing scheme,. *IEEE Transactions on Software Engineering,*, 17(7):725–730, 1991.

[19] Tanaka T. Configurations of the solar wind flow and magnetic field around the planets with no magnetic field: Calculation by a new mhd,. *Journal of Geophysical Research,*, pages 17251–17262, Oct 1993.

[20] Du X. and Zhang X. Coordinating parallel processes on networks of workstations,. *Journal of Parallel and Distributed Computing,*, 46(2):125–135, 1997.

[21] Ma X., Winslett M., Lee J., and Yu S. Faster collective output through active buffering,. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS.02)*, pages 34–41. IEEE, 2002.

[22] Qin X., Jiang H., Zhu Y., and Swanson D. Boosting performance for i/o-intensive workload by preemptive job migrations in a cluster system,. In *Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD03)*, pages 235–243. IEEE, 2003.

[23] Qin X., Jiang H., Zhu Y., and Swanson D. Dynamic load balancing for i/o- and memory-intensive workload in clusters using a feedback control mechanism,. In *Proceedings of the 9th International Euro-Par Conference on Parallel Processing (Euro-Par 2003, Klagenfurt, Austria, Aug.26-29, 2003.)*, 2003.

[24] Qin X., Jiang H., Zhu Y., and Swanson D. Dynamic load balancing for i/o-intensive tasks on heterogeneous clusters,. In *Proceedings of the International Conference on High Performance Computing (HiPC03).*, Dec 2003.

[25] Qin X., Jiang H., Zhu Y., Swanson D., and Feng D. Improved read performance in a cost-effective, fault-tolerant parallel virtual file system (ceft-pvfs),. In *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID.03)*, 2003.

[26] Qin X., Jiang H., Zhu Y., Swanson D., and Feng D. Scheduling for improved write performance in a cost-effective, fault-tolerant parallel virtual file system (ceft-pvfs).,. In *Proceedings of The 4th LCI International Conference on Linux Clusters: The HPCRevolution 2003*, June 2003.

[27] Cho Y., Winslett M., Kuo S., Lee J., and Chen Y. Parallel i/o for scientific applications on heterogeneous clusters: A resource-utilization approach,. In *Proceedings of the 13th international conference on Supercomputing*, pages 253–259. ACM Press New York, NY, USA, 1999.